AUTOMATION IN THE HLA FOM DEVELOPMENT PROCESS

Mr. Robert Lutz
Johns Hopkins University/Applied Physics Lab
Laurel, MD

Mr. Michael Hooks
TASC
Arlington, VA

Mr. Ken Hunt
AEgis Research, Inc.
Huntsville, AL

KEYWORDS
Automation, Process, Tools, OMT, FOM, SOM, DIF

ABSTRACT

A high-level process model has been defined and documented by the HLA program to characterize the sequence of activities recommended to develop and execute HLA federations. Throughout the description of this process model, the potential utility of automated software tools is frequently highlighted. The purpose of this paper is to provide a more detailed investigation as to how automated tools may facilitate the process of developing HLA Federation Object Models (FOMs). The paper will begin with a discussion of the HLA FOM development process, and what key capabilities automated tools may provide in this process. Then, an overview of two existing proof-of-principle FOM development toolsets will be provided, including the design of the toolsets and the major features they provide. Finally, the paper will be summarized by describing how these proof-of-principle toolsets may be leveraged to promote and facilitate the development of more complete automated toolsets to support federation development.

THE HLA FEDEP MODEL

The development of the Department of Defense (DoD) High Level Architecture (HLA) was initiated in response to the DoD Modeling and Simulation (M&S) Master Plan, which calls for a DoD-wide common technical framework which will apply to the full range of potential M&S applications. The objective of the HLA is to facilitate interoperability among simulations and promote reuse of simulations and their components.

The core of the HLA concept is the federation. A federation is a named set of interacting simulations, a common FOM, and supporting runtime infrastructure software, that are used as a whole to achieve some specific objective. The HLA baseline definition has been developed via a set of prototype federations (protofederations), which implemented a diverse set of applications using the initial HLA specification. The experiences of these prototypes have been used to evolve the specification to establish the HLA baseline. Beyond their role as a technical testbed, these HLA protofederations have also been a valuable resource in understanding the process of developing and executing HLA federations. Throughout the HLA prototyping phase, the HLA Object Model Template Working Group (OMTWG) has provided the primary forum for sharing federation development experiences among the protofederations, and for building a baseline process model specification based on the projection of the collective experiences of the protofederations toward future HLA applications.

A graphical representation of the HLA Federation Development and Execution Process (FEDEP) Model is provided in Figure-1. An overview of the FEDEP Model is provided in a separate paper ("A Process View for DIS++", 96-15-007), with a more detailed explanation of the content and relationships between the various subprocesses available through the online HLA Technical Library under the title "HLA Federation Development and Execution Process Model".

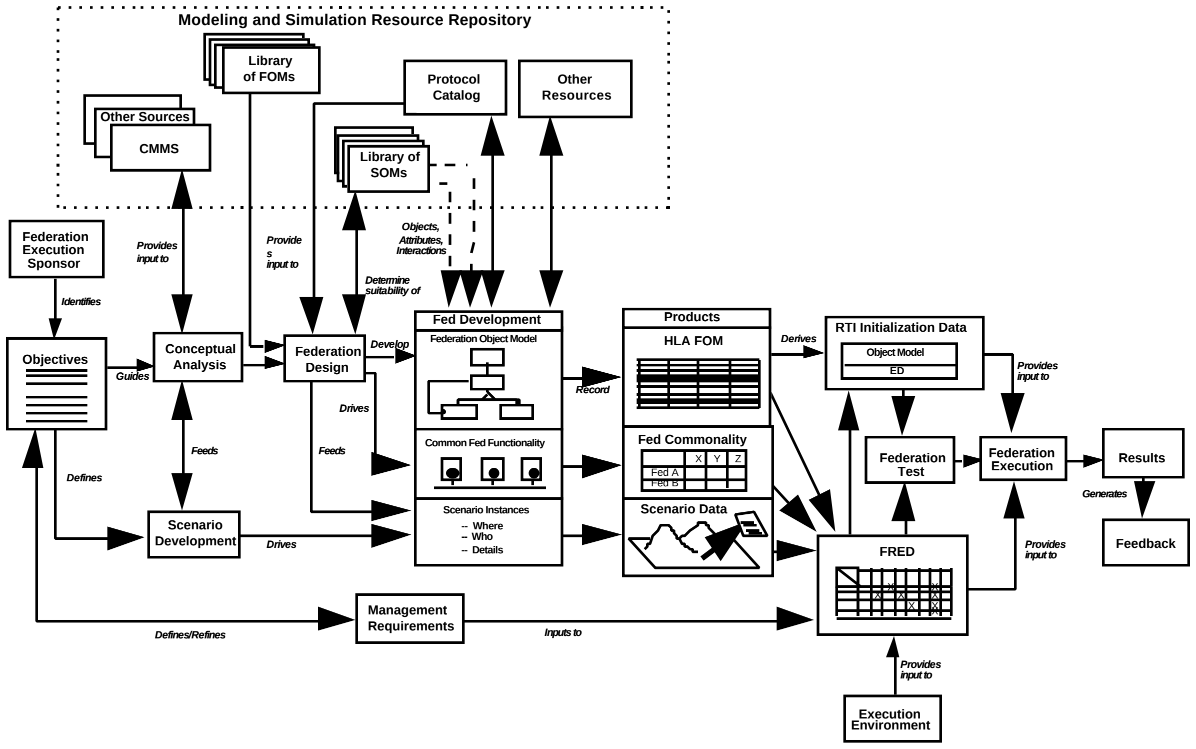

Figure-1. HLA FEDEP Model

The development of the initial HLA federations was, by necessity, a highly manual process due to the general absence of automated tools to support the HLA prototyping phase. Although this was considered temporarily acceptable due to the limited nature of the experimentation, several of the "lessons learned" from this experimentation expressed concern about the scaleability of the overall federation development process to large applications (eg., STOW). In order to address these concerns, new emphasis has recently been given to the need to automate certain aspects of federation development. During construction of the FEDEP Model, several opportunities for introducing automation into the federation development process were both identified and documented. A notional subset of the full range of HLA tool requirements is provided in Table-1.

In order to begin to address these longer-term needs for automation, the Defense Modeling and Simulation Office (DMSO) has recently initiated two proof-of-principle efforts to develop supporting toolsets for federation construction. The initial focus of these toolsets are on support for FOM development, which was found by the HLA protofederations to be particularly labor intensive to accomplish manually. The decision to build two different versions of a FOM Development Toolset (FDT) was based on a desire to:

- demonstrate how different implementation strategies can be utilized to build a common, core set of required functionalities,
- stimulate creativity in defining supplemental functionalities that may be appropriate in particular communities-of-use, and
- demonstrate the ability to exchange FOM data among different FDTs via a common Data Interchange Format (DIF), based on the information requirements described in the HLA Object Model Template (OMT).

TOOL	REQUIREMENT
CMMS Tool	Access authoritative DoD meta-models of real world behavior and define the required subset of these models required for achieving the scope defined by the federation's Objectives Statement.
Scenario Development Tool	Accept definition of mission space from CMMS Tool. Allow scenario developer to create scenario by deploying instances of objects from the mission space, enforcing rules defined by CMMS.
Exercise Planning Tool	Accept scenario from Scenario Development Tool. Accept FOM from FOM Development Tool. Allow the user to map scenario elements to FOM objects, and instances of FOM objects to specific federates.
FOM Development Tool	Access MSRR libraries of FOMs and SOMs. Aid federation developers in building FOM. Perform consistency checking. Output RTI Initialization Data (RID) for federation execution.
Federation Controller	Instantiate the federation. Provide controls for starting, pausing, and stopping the federation.
Scenario Monitor	Provide visualization of scenario during execution.
Post-Processing Tool	Read runtime data logged by federation. Perform data reduction and calculation of MOE/MOP metrics.

Table-1. HLA Tool Requirements

The organizations chosen to develop the proof-of-principle FDTs were Aegis Research, Inc. of Huntsville, AL and TASC of Arlington, VA. The next two sections summarizes both the design and the major features of both of these toolset development efforts.

TASC FDT

HLA FOMs and Simulation Object Models (SOMs) are two of the key factors in determining simulation interoperability. These object models define the Federation's or a particular simulation's standards for class structure, class inheritance hierarchy, simulation interactions, object relationships, and programming data types. Using one of the most prevalent FOM/SOM specification tools in the HLA prototyping phase, MS Excel spreadsheets, to fully specify all the elements in the FOM or SOM has proven to be labor intensive and problematic. A spreadsheet does not adequately support either object-oriented design methodologies or the CASE tool change propagation necessary to build an internally consistent object model. Automated tools are clearly required to support the wide range of users and their diverse needs and requirements for creation and maintenance of HLA FOMs and SOMs.

Design

TASC's technical approach to the FDT is based on the following key design goals:

Common API: Build an extensible architecture with a common Application Programmer Interface (API). The common API allows our own developers, as well as other tool vendors, to augment the FDT functionality and change sections of the system's implementation without affecting the end users' interface. This design discipline helps protect tool investment.

Platform Independent GUI: Develop a Graphical User Interface (GUI) that is architecture neutral. The GUI implementation should be independent of the host machine and operating system. This capability provides users with the flexibility to access the FDT from a either a MAC, PC, or any Unix platform using a GUI with a common "look and feel".

OMT DIF Compatibility: The FDT should interoperate with the Object Model Template (OMT) Data Interchange Format (DIF), a standard file exchange format for both FOMs and SOMs.

Database Independence: The users' local object model repository should not depend on a specific COTS database. The FDT should use an API to the database, rather than services that are restricted to a specific database. Users should not be required to purchase machines and database products to use the FDT. The FDT should accomodate the users' existing resources as much as possible.

Distributed Access: The FDT architecture should support multi-user distributed access. The FDT should be available over the network to multiple users on different platforms for viewing and modification. This capability is necessary for FOMs and SOMs that require a coordinated development effort from multiple individuals geographically separated.

Configuration Management: FOMs and SOMs should be configuration managed. This level of accountability is required because of the FOM's role in determining simulation interoperability.

Based on these design goals, TASC has chosen to integrate two emerging object technologies, CORBA and Java, into the FDT architecture. CORBA is an Object Management Group (OMG) specification that defines a set of services and interfaces that compliant ORB vendors must provide. The vendors' implementation of the CORBA standard provides the Interface Definition Language (IDL), client and server libraries, meta-data repository, and run-time support for developing distributed applications. Using Iona's CORBA implementation, Orbix, allowed TASC to meet it's design goals for an industry standard API, internet and intranet distributed access to the model repository, and multi-user support. Orbix also provided TASC with the language mapping required to develop our architecture neutral GUI. Orbix has a language mapping for Java.

Java is an object-oriented language developed by Sun Microsystems. Java, similiar in syntax to C++, is designed to be robust, architecture neutral, portable, and multi-threaded. Java source code and the compiled binary files are platform independent. The source files are compiled into bytecodes, a set of instructions generated according to a virtual machine language specification. The same set of generated bytecodes can be interpreted and executed on almost every available platform and operating system. The combination of Java and CORBA provides users with a network accessible, platform independent GUI.

Architecture/Features

The FDT architecture is separated into three primary subsystems:

- Object Model Management System (OMMS)
- Java Graphical User Interface
- Extended Toolset

The OMMS is a CORBA accessible server that manages access to the FOM and SOM repositories and controls their creation, deletion, and modification. The OMMS also supports versioning of individual FOMs and SOMs and change propagation. When object model elements are modified, their relationships to other elements in the object models are verified. The change propagation layer ensures the integrity of the model repository and notifies the user of any additional changes that are made to other model elements to maintain consistency. For example, if an object class is deleted, all interactions, associations, and compositions are checked for dependencies. If any of these elements were using the deleted object class, corrections are made and the user is notified of the changes. Other OMMS features include network accessibility and the separation of the system interface and it's implementation. The OMMS is accessible from any machine on the network, but the ability to launch or activate the server is controlled based on unique user and group identifiers. Since the API is written in IDL, future changes in the OMMS are isolated from the users. The implementation of certain services can be modified or an ORDBMS or ODBMS can be added without affecting the users or client tool developers. This is extremely important when considering code reuse and protecting tool investment.

The Java GUI provides the user with the ability to open, modify, and version FOMs and SOMs. The GUI supports a a graphical representation of the elements in the object model and displays a comprehensive view of these elements. There are numerous navigational capabilities with multiple views into the object model. Views exist for the top level object model, class structure, class inheritance hierarchy, interactions, and the various constructed data types (structures, enumerations, and unions). The GUI provides services to aid the user in building and maintaining large object models. The top level view and tabbed panels allows the user to easily navigate and display any element in the object model. Most the views support a control panel to index through all the elements of the selected type. Users can also use the inheritance view to navigate through the full class inheritance hierarchy. One of the major challenges in building a large object model is tracking created attributes in the class inheritance hierarchy. With larger models, duplicating attributes can complicate the design process. The FDT resolves this issue by displaying, for each class, all the leaf node attributes and the inherited attributes. For any of the leaf node attributes, the user can also "promote" the attribute to a base class.

The extended toolset presently includes three code generators: an IDL/C++ code generator, a DIF generator, and a RID generator. The IDL/C++ code generator ensures consistency between the data types generated in the object model and the programming data types used in the simulation. The DIF generator reads and writes object model representations both from and into the OMT DIF. Finally, the RID generator enforces consistency between the RID file and the FOM.

AEGIS RESEARCH FDT

The Aegis FDT is a Microsoft Windows 95/NT based application. The FDT is designed to be intuitive to use for those experienced with the Microsoft Office application suite. In addition to providing an object model editing environment, the FDT includes a built-in interface to the Modeling and Simulation Resource Repository (MSRR), as well as interfaces to commercial, off-the-shelf (COTS) computer-aided software engineering (CASE) tools.

Design

The FDT is developed in Visual C++, using the Microsoft Foundation Class (MFC). This development environment provides its own methodology for the overall application framework, and provides a rich set of software objects for building highly standardized user interfaces. The MFC is built around the abstract concepts of documents and views. In spite of the simplified name, a document is not necessarily a single file. Although the abstraction allows a document to be a simple text file, it could also represent a collection of files, or even a database. The major design elements of the AEgis FDT include:

Multiple Document/Views: The FDT is built around the MFC concept of a multiple document / multiple view application. The contrast between single document and multiple document applications is evident in the differences between the Windows Notepad application, a single document application that requires you to close an open file before opening another, and the Microsoft Word application, which is a multiple document application that can have many files open simultaneously. For the FDT, this translates into support for editing multiple FOMs and SOMs simultaneously. The window title bars of the various views indicate the object model's name, as well as the view's name. The standard concepts of Cut, Copy and Paste allow information to be moved between the different object models.

The FDT also takes advantage of the MFC support for the concept of multiple views into a document. Again, to contrast between single and multiple views consider two widely known commercial applications: Microsoft Word, while supporting multiple documents, is a single view application. There is only one way of looking at the document. Intuit's Quicken, however, supports multiple views. It provides a variety of ways of looking at and editing a bank account.

Standardized User Interface: The FDT utilizes several of the MFC standard menu and toolbar items. The File, Edit and Window menu items should be intuitive to navigate for anyone who is comfortable with Windows or Macintosh applications. The File menu provides support for creating new object models, opening any of the supported file formats (FDT, DIF, RID, etc.), closing an object model, and saving modifications to the object model. The print and print preview is also available from this menu, as well as a continuously updated list of the most recently opened object models. The Edit menu provides the Cut, Copy and Paste support. The Window menu provides support for arranging opened windows, as well as maintains a list of all open windows to assist the user in finding a particular window when many are opened. The standard toolbar shortcuts for New, Open, Save and Print are available on the FDT toolbar. The FDT also defines its own extensions to this toolbar.

Property Sheets: The FDT provides views for each of the HLA OMT tables. The FDT relies heavily on the Windows 95 style of right-mouse activated menus. From the Class View window, for example, the right mouse button menu provides access to the property list of the selected class, as well as commands for inserting new classes. Opening the class property list allows the user to modify the definition of the class, or to define the properties of a new class. The tabbed index property list contains property sheets for general information, such as class name, glossary definition and comments, as well as separate sheets for class attributes and class components. Modifying the properties of a class will update any affected view of the object model.

Reusable Components: In support of the overall tool suite for HLA, the FDT provides two key reusable software components: the DIF file parser and the OMT Object Model class library. The FDT reads the DIF and RID files via parsers developed from using the UNIX standard lex and yacc utilities. The lex utility uses an input specification that describes the fundamental pieces of information that exist in input files (tokens, in lex terms), and generates C code for a lexical analyzer to break up an input stream into tokens. The yacc utility uses an input file that consists of grammar rules describing patterns of the tokens that it should parse, and generates C code for a parser. The lex and yacc input specifications developed for the FDT parsers are readily available for other tools in the HLA tool suite that need to read these files. These parsers have been tested on both UNIX and Windows 95 platforms, and are independent of any platform constraints. To aid developers who are not proficient in lex/yacc parsers, the yacc parser makes callbacks to a set of C functions as input patterns are recognized. These C functions can be modified and used to construct representations of object models in any C or C++ application.

The AEgis FDT is built on a configuration of several object libraries. One of these, the OMT Object Model class library, maintains an in-memory representation of the object models being edited (i.e. FOMs, SOMs). To support the FDT and potentially other HLA tools, this in-memory representation has been developed as a Dynamic Link Library (DLL). This library contains an "Object Model" class definition, as well as all the necessary supporting classes (Class, Interaction, Data Type, etc.) needed to represent an HLA object model in memory. These classes define a useful set of methods for searching, traversing, manipulating, and reading and writing HLA object models. These C++ classes have been designed to function independently from the platform-dependent user-interface code of the FDT application to maximize the potential for reusing these software components in other HLA tools.

Features

The following provides a summary of the major features currently supported by the AEgis FDT:

MSRR Interface: The FDT uses the Microsoft ActiveX control for importing remote FOMs and SOMs to the local workstation from the http-accessible MSRR. The MSRR contains archives of FOMs and SOMs to allow new federations to leverage off the engineering investment of previous federations. The ActiveX software object essentially provides the user with a mini web browser from within the FDT. The imported files are transferred across the http interface as an HLA Data Interchange Format (DIF) file.

Format Translation: The FDT File menu's "Open…" command presents the user with the standard File Browser Dialog common to Windows 95 applications. The FDT File Browser will support several file types. The default, of course, is the native FDT file type. The "File types" popup control allows the user to open DIF files as well. When an object model is loaded from an imported DIF file, modifications will be saved back to the DIF file by default, but the "Save As…" feature allows the file to be translated to the native FDT format as well.

In a similar manner, object models created in the native FDT file format can be translated to a DIF file. The "Save As…" menu option under the File menu allows the file to be saved as FDT or DIF. It should be noted that this transfer between FDT and DIF file should be used with care. The native FDT file format is essentially "owned" by the FDT, and may contain information extensions not supported by the DIF. These extensions would essentially be lost in the translation to DIF.

Another file type that can be written is the RTI Initialization Data (RID) file. The RID is the subset of the FOM data actually used by the RTI during runtime. Just as the DIF contains a subset of the information of the native FDT file, the RID is a subset of the DIF. And although translation to the RID format results in a significant loss of information from the native FDT format, the FDT can open a RID file as an object model specification.

CASE Tools: The FDT attempts to bridge the gap between the HLA OMT and true object-oriented modeling tools by supporting the interchange of information with commercial off-the-shelf (COTS) object-oriented design (OOD) CASE tools. The FDT connects directly to Object Server database of Paradigm Plus for importing an object model definition into the FDT. Also, the FDT installation package includes a Rose script file that allows HLA specific information to be attached directly to a Rose model. With this script extension, much of the HLA OMT specification can be captured directly within the Rose object model.

Help System: The FDT implements the advanced help system expected of Windows 95 based applications. A variety of help subsystems are available, including the Windows 95 concept of electronic books. The FDT contains access to three volumes of documentation: The FDT User's Guide, FDT Reference Manual, and the HLA OMT Documentation.

The User's Guide describes a walk through of the definition of an object model using the FDT. It introduces FDT-specific terms and concepts as they are needed, and contains references as hypertext links to the other electronic volumes when appropriate. The FDT reference manual provides definitions of FDT specific terms and concepts. The HLA OMT document is included to provide the user rationale and context for the FDT.

In addition to the electronic books, all graphical controls within the FDT have context-sensitive help. These pop-up descriptions of control use and effect generally include hypertext links to related information in the electronic books. A help index contains an alphabetical index of FDT terms and commands, and keyword searches can be made of all the electronic books available, or selected subsets of them.

Printing: The FDT supports printing the object model views. Since each FDT view corresponds to one of the HLA OMT tables, the FDT essentially prints in the HLA OMT format. The print dialog allows the user to choose between printing the currently selected view, or all views for the currently active object model. The Print Preview menu option allows the user to adjust scaling and margin information.

Portability: The Visual C++ development environment contains several paths for application portability. Both Visual C++ and the MFC are available for the Macintosh System 7 operating system. Some slight modifications are generally necessary to the application's Help file specification and resource definitions to port the code from Windows to the Macintosh, but this is a trivial effort compared to the overall development of the tool.

A variety of options exist for porting from Windows to a UNIX platform. Motif-based implementations of the MFC are available for porting the application to a true UNIX application. Alternatively, various Windows emulators exist that allow the binary image to run directly without porting.

SUMMARY

It is currently envisioned that it is through the introduction of automated toolsets (like those discussed above) that the HLA will evolve to be a viable and desireable paradigm for development of large exercises. The significant gains in efficiency (and corresponding reductions in resource requirements) achievable through automation is expected to stimulate a widespread commercial marketplace for HLA toolsets. Not only will this proliferation of tools continue to simplify and streamline the process by which HLA federations are built and executed, but will also simplify the process by which new organizations may join and participate in HLA federations.

These two proof-of-principle efforts provide an initial demonstration of the types of capabilities which are possible to develop to support the HLA FOM development process. It is believed that these initial FDTs provide an adequate baseline capability which can be leveraged by either government agencies and/or commercial firms to develop a more complete set of FOM development support capabilities based on user needs. It is also hoped that this initial demonstration of potential capabilities provides a stimulus for organizations to begin designing, developing, and experimenting with other aspects of HLA automation, such as conceptual modeling and execution support. The long-term acceptance and success of the HLA will be highly dependent on this automation.

REFERENCES

All references are available for download via the home page of the Defense Modeling and Simulation Office, site address http://www.dmso.mil.

[1] Under Secretary of Defense for Acquisition and Technology, “Department of Defense Modeling and Simulation Master Plan, DoD 5000.59-P,” October 1995.

[2] Defense Modeling and Simulation Office, “HLA Federation Development and Execution Process Model, Version 1.0,” August 1996.

[3] Defense Modeling and Simulation Office, “HLA Object Model Template, Version 1.0,” August 1996.